INDAGACIONES SOBRE LOS PROCESOS DE DESARROLLO Y "MANTENIMIENTO" DE SOFTWARE: UN ENFOQUE 3-P

Por F. Saéz Vacas.
Catedrático de Cibernética y Ordenadores de la Escuela Técnica Superior de Ingenieros de Telecomunicación de Madrid.

1. LAS TRES PES: PROBLEMA, PRODUCTO, PROCESO.

Se ha convertido en costumbre (justificada, desde luego) atribuir genéricamente al software un conjunto de defectos: costes elevados, infiabilidad, intransportabilidad, etc. También sus procesos de desarrollo y mantenimiento tienen bien ganada reputación como infatigables manaderos de dificultades y conflictos.

Bastantes de estos rasgos se deben al destino único e irrepetible[1] de muchas piezas de software, característica de uso que es, probablemente, una de las causas sociales más importantes de falta de estímulo para el perfeccionamiento profesional en este terreno. Como área, además, hay que reconocer que se encuentra en su infancia, lo que acentúa la situación de incongruente rigor a que se ve sometida, al exigírsele a cada pieza de software y, por ende, al software como especie, una extremada perfección (hablando en términos profesionales).

En todo caso, la problemática del software a la que nos estamos refiriendo se pone de manifiesto más y más a medida que crece la envergadura de la pieza. Por fijar ideas y con todas las salvedades y matizaciones que cabría hacer, el lector puede imaginar que las consideraciones que siguen serían plenamente válidas para una aplicación informática cuyo plazo de desarrollo por un equipo humano adecuado igualase o superase el año. El efecto de escala en el esfuerzo se ilustra muy plásticamente por la analogía comparativa de los puentes, según Fox (1982), al que citaremos más de una vez.

	Pasarela en un parque	Puente Verrazano
Requisitos	1 día	1.825 días
Diseño	1 hoja papel	almacén de papel
Plan de material	1 hora	1.460 días
N° de personas	2	5.000
Construcción	3 días	1.825 días
Documentación	1 día, 5 hojas papel	555 días, 500.000 hojas de papel
COSTE TOTAL	$ 1.000	$ 300.000.000

Figura 1. Efectos de escala en el esfuerzo (Fox, 1982)

Conviene distinguir dos vertientes en el software: el objeto en sí, o pieza de software propiamente dicha, y el proceso del software, entendido como conjunto de actividades orientadas a producir o conservar ese objeto. La comparación de la figura 1 entre la pasarela del parque y el puente Verrazano da una idea luminosa de hasta qué punto puede ser amplio el rango de variabilidad de los procesos y diferentes las técnicas.

Se está imponiendo internacionalmente el término de Ingeniería de Software como referencia del conjunto de técnicas y tareas para la "especificación, diseño, implementación, prueba y explotación de programas de computador" (Zelkowitz, 1979). En esta definición, que hay que puntualizar y ampliar, se presentan indisociados (e indisociables) el proceso (especificación, diseño, etc.) y el objeto o producto (programas)[2]. La palabra 'ingeniería' indica ya la pretensión del enfoque, aún cuando, por el momento, su grado de evolución diste mucho

1. 'Unico e irrepetible' significa 'preparado para una sola instalación', sin los cuidados necesarios para una adecuada transferencia. Además, hay que tener en cuenta que, según estadísticas, alrededor del 85% de las piezas de software que se desarrollan solamente se ejecutan una vez. Este grado de irrepetibilidad da una idea de cuanto se repite el esfuerzo de hacer las mismas cosas.

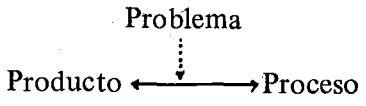
2. Software es un conjunto de programas que interactúan. Se acostumbra incluir en el software documentación y procedimientos. Al vocablo 'producto' no se le da aquí el sentido calificativo de software-producto, con el que se conoce a un software preparado para ser vendido en el mercado a muchos usuarios.

de estar a la altura de otras ingenierías (mecánica, eléctrica, química...).

El producto, cuya sustancia es la información, posee contenidos tan variados como quepa imaginar, puesto que es aplicable a todos los dominios de problemas del quehacer humano. Se elabora por procesos mentales extendidos en una gama muy amplia de diversos niveles y características intelectuales, dentro de un entorno tecnológico de rapidísima evolución. Al producto se le exige un funcionamiento muy preciso, se le somete a un permanente control de calidad y, en muchos casos, es obligado adaptarlo contínuamente a los cambios de definición de la aplicación. Tanto su proceso de producción como de utilización van siendo progresivamente sometidos también a control económico.

Intuitivamente, se percibe que la naturaleza del problema que se quiera resolver (desde la perspectiva de su representación lógica o modelación informacional) y la propia naturaleza de la solución software son factores de primerísimo orden de importancia.

Aunque esté por hacer una tipología de los problemas en lo que se refiere a su caracterización informática y también de las soluciones software según las propiedades configuradoras de su proceso de desarrollo y de su uso, es indiscutible la existencia de una ligadura entre problema, producto (software) y proceso. Por ahora, la represento por el ideograma siguiente:



Con él expreso simplificadamente[3] una jerarquía de ideas, mejor ilustrada por el diagrama de la figura 2. Ambos gráficos se irán describiendo en este y en sucesivos apartados.

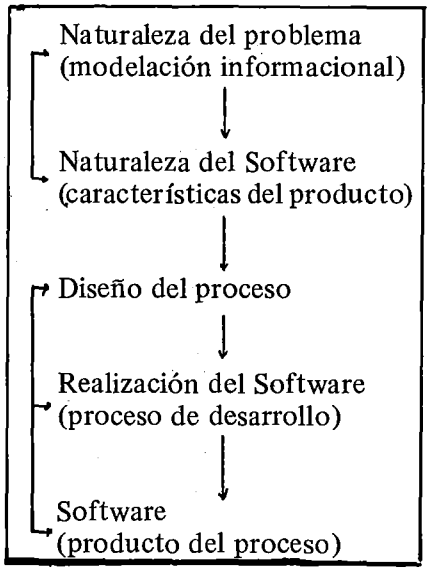


Fig. 2. Jerarquización circular de las tres pes.

Resaltaré ahora algunos rasgos generales y fundamentales del proceso. En primer término, el proceso es, en su totalidad, un conjunto ordenado de procesos mentales, lo que introduce un factor distintivo inmenso de índole psicológica y social: el factor humano, en el que no se entrará aquí.

Desde un punto de vista organizativo, los niveles de abstracción, tipos de entidad, clases de decisión y problemas que se manejan desde el principio al final del proceso ofrecen suficientes diferencias para poderlo descomponer en una secuencia de cierto número de etapas aceptablemente delimitadas. Y esto es lo que se ha hecho, dando lugar al desglose llamado ciclo de vida del software. Como tantas veces ocurre, un concepto se difunde, se aplica con mayor o menor rigor, acierto o fortuna, pero no se profundiza lo necesario en sus condiciones de aplicación, con lo que, al riesgo de malutilizarlo, se añade el peligro de banalizarlo.

El ciclo de vida se fundamenta en la noción de cambio, de crecimiento, de adaptación, circunstancia que no afecta o no afecta por igual a todos los problemas que puedan plantearse. Tal cosa me lleva a recordar una muy interesante taxonomía[4] de los programas (en realidad, planteamiento del problema), según los índices previsibles de cambio. Los programas son, así, de uno de estos tres tipos: S, P o E, siendo el último el que mecaniza actividades humanas o sociales y se integra en el mundo real que modela. Por eso, sus fuentes de cambio son muy variadas, como expresa la figura 4: discrepancias entre resultados y problema, insatisfacción sobre la forma de los resultados, evolución del mismo problema. Sólo los tipos P y E están sujetos a cambio y, por tanto, a

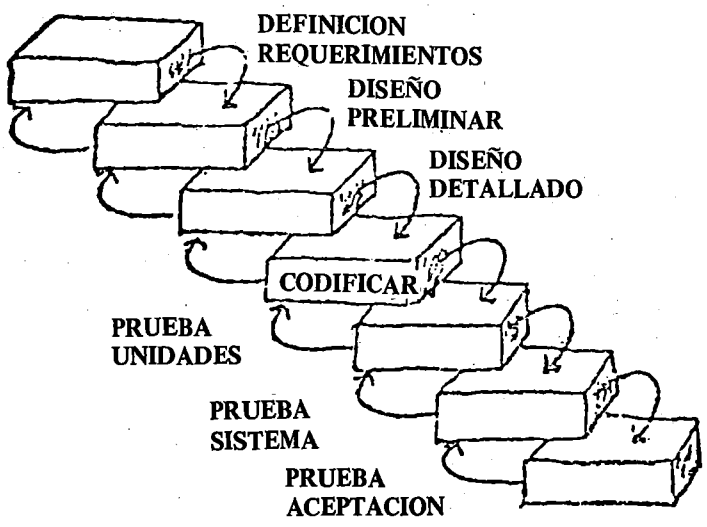


Fig. 3. Ciclo de vida del software (falta por representar la etapa de mantenimiento).

3. Un esquema ampliado a cinco pes se incluye en una comunicación titulada. "Some framework ideas for Software Engineering Education", que el autor ha enviado al Comité de Programa de la 7th International Conference on Software Engineering (Florida, marzo 1984).

4. La clasificación parece que se debe a Turski, pero es Lehman quien la ha descrito en un sugestivo artículo sobre programas, ciclos de vida y leyes de evolución (Lehman, 1980).

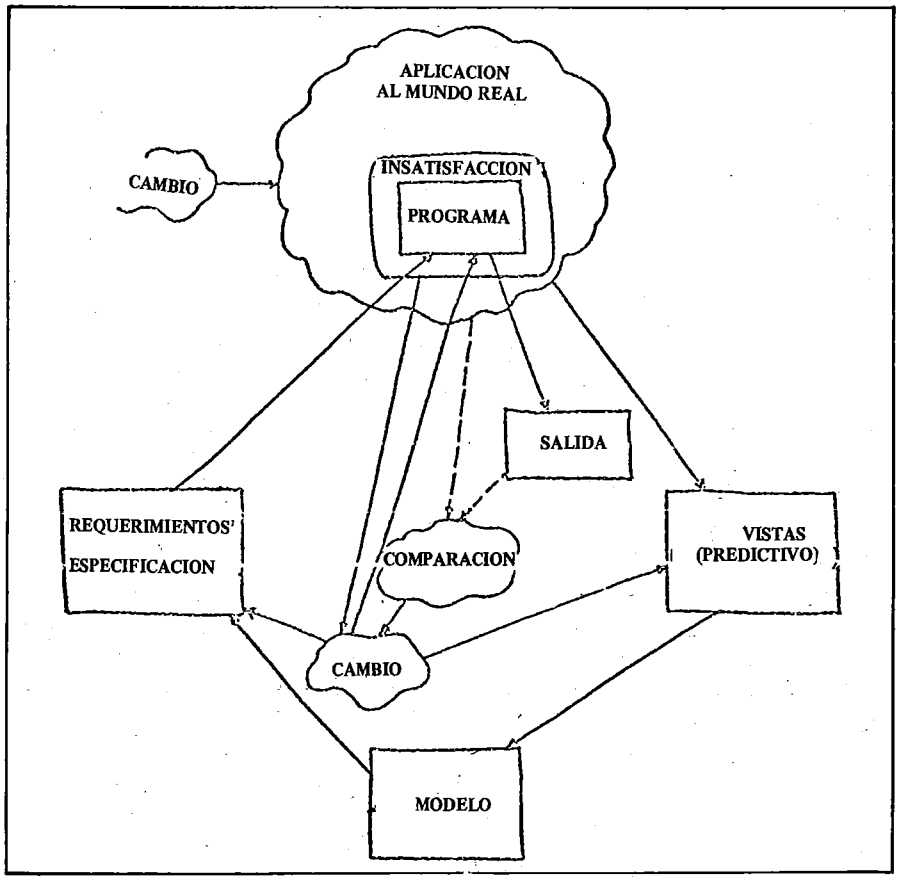


Fig. 4. E-programas (Lehman, 1980).

evolución, mientras que el tipo S pertenece al dominio de lo formalmente especificable o de lo conceptualmente estático.

Desde luego, esta clasificación presenta más interés teórico que práctico, pero tiene la virtud de poner de relieve una cualidad inmanente al problema, su tasa de cambio, que juega un papel primordial en las características del software y del diseño del proceso.

La primera consecuencia es que un problema cuya tasa de cambio sea nula, requerirá un proceso sin mantenimiento. Propiamente no hay ciclo de vida. Ejemplos, hay muchos: la resolución de un método matemático determinado, el control del lanzamiento de una concreta nave espacial, etc.

A medida que la tasa de cambio crece, aumenta la importancia de la fase (mal llamada) de mantenimiento, hasta el punto de que, como se verá en el próximo apartado, el ciclo de vida debe asumirse indefectiblemente y desde el principio como dominado por dos grandes fases. Es el primer enfoque del proceso.

Un segundo enforque del proceso procede, en mi opinión, de otra cualidad perteneciente a la naturaleza del problema en cuanto al grado de nitidez con que éste es percibido por los diseñadores: su complejidad o borrosidad. Cuando esta cualidad se manifieste el ciclo constará de tres grandes fases.

2. LOS ENFOQUES DEL PROCESO. PRIMERA Y SEGUNDA APROXIMACION.

Una incial aproximación del proceso de arriba abajo nos subdivide éste en dos partes, claramente diferenciadas en el tiempo y en el propósito. La primera es el proceso de desarrollo o producción del software y la segunda, el proceso de mantenimiento[5] cuya finalidad es corregir deficiencias y conservar, mejorar o actualizar las propiedades del objeto. Partimos de la hipótesis de la necesidad de cambio.

Es indudable que ambas partes, aunque separables en ciertos aspectos, constituyen un proyecto global que abarca ese objeto durante toda su vida, desde su creación hasta su muerte (de ahí el nombre de "ciclo de vida").

A esta partición no se le ha extraído en la práctica el jugo adecuado, precisamente porque se le han aplicado en exclusiva criterios de separación y no de

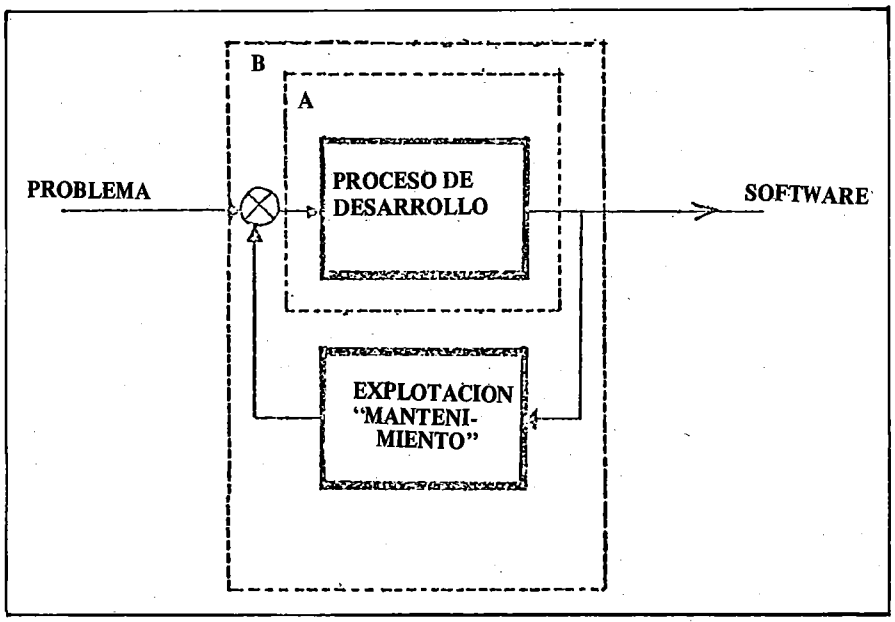


Fig. 5. Ciclo de vida, como sistema cibernético.

integración, siendo así que ambas dimensiones son consustanciales. La experiencia nos inunda de ejemplos demostrativos al res-

5. Denominación universal, muy poco afortunada, que parece excluir la idea de evolución. La figura 5 expresa el mantenimiento como proceso de corrección (si el problema no cambia) y como proceso de adaptación o evolución (si el problema o su definición cambian).

pecto y la bibliografía se mueve habitualmente en esa misma línea, aunque hay excepciones notables, por ejemplo (Fox, 1982).

Puede representarse el ciclo de vida de otra forma gráfica, muy didáctica, por analogía con los sistemas cibernéticos.

No es lo mismo enfocar el proceso de desarrollo de software como si no existiera el proceso de mantenimiento (bloque A de trazos de la figura 5) que concebirlo desde el inicio en forma integral (bloque B). Tan secilla figura contiene aspectos fundamentales que diferencian los resultados de uno y otro enfoque. El enfoque B considera el asunto como un sistema dinámico, en donde el problema es algo que cambia con el tiempo, lo que pone de manifiesto la evolutividad natural del software, alimentándose contínuamente el proceso con las discrepancias entre la solución software y las necesidades reales del problema (el símbolo representa un comparador). Anular recurrentemente[6] dichas discrepancias constituye el objetivo básico del proceso, desde la perspectiva B. Es decir, el proceso se diseña y se optimiza para conseguir este objetivo que se acaba de señalar. Bien sabido es que los diseños dependen radicalmente de las metas fijadas para ellos. Así pues, un factor crítico es utilizar el enfoque B o el A, y afirmo que, en la práctica histórica de la informática no se ha incluido realmente (sí verbalmente) la etapa de mantenimiento en el bloque llamado "proceso" de la figura 5; en otras palabras, se ha hablado mucho del ciclo de vida del software pero se ha utilizado poco en el diseño del proceso, siendo ésta una de las causas principales de las dificultades del software. El bloque "proceso", en el enfoque B, considera la actividad de mantenimiento en forma distinguible, pero no separable del diseño, perspectiva que, si siempre es importante, en muchos casos es vital (recuérdese el concepto de "tasa de cambio").

Un conocido experimento de programación (citado en Boehm, 1981) muestra la gran sensibilidad de las propiedades reales del producto (software) ante los criterios conformadores de su proceso de desarrollo. Se encargó un mismo trabajo a cinco equipos distintos y a cada equipo se le fijó un criterio optimizador: 1) completar el trabajo con el menor esfuerzo, 2) minimizar el número de sentencias fuente, 3) minimizar ocupación de memoria, 4) maximizar la claridad del programa, 5) maximizar la claridad de las informaciones de salida.

Al cuadro de la figura 6 se le podrían extraer diversos comentarios jugosos, a pesar de que el objeto del experimento difiere considerablemente de los objetos-software que se están considerando aquí.

Pero nos interesan conclusiones generales. La conflictividad de los criterios se hace evidente en la dispersión de los resultados. Siguiendo la línea de mi razonamiento, es posible aglutinar tales resultados en dos grandes paquetes, de acuerdo con dos macrocriterios: A) el proceso está orientado al desarrollo; B) el proceso está orientado al ciclo de vida.

Si entendemos que, en la figura 6, los tres primeros objetivos guardan relación con el desarrollo sin más y los dos últimos manifiestan una cierta preocupación por el ciclo de vida, el experimento arroja unas correlaciones contrapuestas que apoyan el argumento de una forma general: ambos enfoques aparecen como antagónicos, lo que explica muchas cosas.

Profundizando ahora en el primer enfoque, se perciben alternativas varias. El proceso tipo A se orienta a: A_1) la máquina; A_2) la productividad (número de sentencias por unidad de tiempo y coste). Dentro de este último grupo se practica o puede practicarse una nueva distinción: $A_{2.1}$) orientación al coste económico; $A_{2.2}$) orientación al tiempo o plazo de entrega (al disminuir los plazos de entrega se disparan los costes, como ha demostrado Putnam, entre otros). Es decir, que, adoptando el enfoque A, que es grandemente contradictorio con el B, no se está por ello exento de contradicciones, ya que una orientación del proceso por criterios de máquina (calidad medida en términos de memoria, número de sentencias, etc.) se opone a la productividad (generalmente se fija el tamaño del software y se pretende minimizar el coste económico o el tiempo de desarrollo, entre confusas definiciones de calidad), y viceversa.

Escala de resultados (1, lo mejor)					
Criterio de desarrollo	Menor esfuerzo	Menor nº Sentencias	Memoria Mínima	Máx. claridad progra.	Máx. claridad salidas
Menor esfuerzo	1	4	4	5	3
Menor nº sentenc.	2 – 3	1	2	3	5
Mínima memoria	5	2	1	4	4
Máx. claridad progr.	4	3	3	2	2
Máx. claridad salida	2 – 3	5	5	1	1

Fig. 6 Experimento sobre conflictividad de criterios en el desarrollo del software (Weinberg, 1974), citado en (Boehm, 1981).

6. En caso contrario, carecería de sentido hablar de ciclo de vida.

La situación es compleja. La resumo, aunque más adelante reorganizaré nuevamente la totalidad de los argumentos.

1. Proceso y software son indisociables. El software depende del proceso puesto que es su producto y el proceso depende del producto, no sólo en cuanto que la adaptación del producto obliga a activar el proceso en permanencia sino porque las características propuestas para el software condicionan las técnicas elegidas en el proceso (véase fig. 2).

2. Se puede optar entre un proceso tipo A y un proceso tipo B, éste último vinculado explícitamente con la necesidad de evolutividad del producto. Los enfoques son contradictorios entre sí, en términos de coste y en términos de un cierto número de criterios de calidad. Todos los datos señalan la manifiesta importancia de la fase de mantenimiento (véase, en figura 7, gráfico (demasiado conocido) de distribución histórica de costes, que presumiblemente es aplicable a aplicaciones informáticas afectadas por apreciables tasas de cambio).

3. No existe criterio único de calidad. En último extremo, la fijación de una meta de calidad[7] del producto, expresada por un complejo de propiedades, orientará definitivamente el diseño del proceso. Lógicamente, ha de buscarse el máximo grado de coherencia en esta elección, dentro de los condicionantes generales recogidos en el segundo punto de este resumen. En todo caso, la calidad tiene que ponerse en congruencia con parámetros típicos del proceso (coste, tiempo, técnicas disponibles), en donde determinadas limitaciones en el coste y el tiempo desempeñan una función constrictora.

7. Velocidad de cálculo, ocupación de memoria, tiempo de respuesta, mantenibilidad, fiabilidad, transportabilidad,...

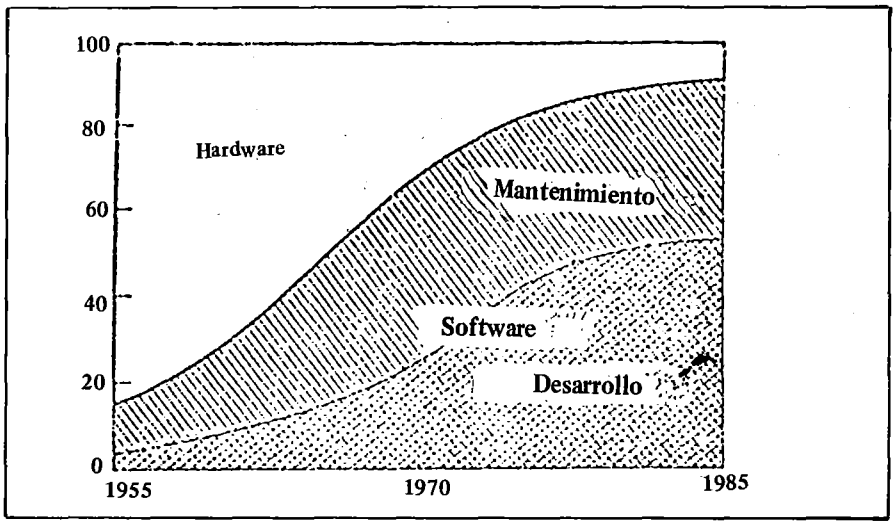


Fig. 7. Costes relativos (Boehm, 1981).

4. Sin entrar en detalles, la prevalencia del enfoque A en la práctica habitual de la informática ha llevado a entronizar una perspectiva cuantitativa productivista en términos de LOC o SF (Líneas de Código o Sentencias Fuente) por unidad temporal de esfuerzo humano. La calidad, cualquiera que ésta sea, queda enmascarada y muchas veces desaparece pulverizada por exigencias de coste o de plazos.

Hay quien ha comparado el software con un rompecabezas y es una acertada metáfora, por muchas y variadas razones. Sin embargo, esta metáfora sólo refleja tímidamente la realidad de muchas ocasiones. En efecto, el software es un rompecabezas, pero un rompecabezas en el que hay que definir el dibujo/modelo, los colores, recortar las piezas, juntarlas y empezar de nuevo con otro dibujo. La dificultad de establecer nítidamente el dibujo es, en parte, un atributo del problema (su complejidad) y, en parte, un atributo del diseñador; la interrelación de ambos atributos plantea una cualidad esencial, no cuantificable, del proceso, que es su grado de borrosidad.

Dependiendo de la borrosidad (complejidad problema-diseñador), los primeros pasos del proceso de desarrollo oscilan entre lo meramente repetitivo o rutinario y la actividad intelectual más profunda y creativa, que consiste en penetrar por la bruma para atrapar las líneas de la solución, el modelo, el sistema, el dibujo. En este trabajo se apoya el resto del trabajo. De ahí la trascendencia del grado de borrosidad que, a mi juicio, puede dar señas propias de identidad a una fase cuya personalidad haría buenos o no muchos de los estudios, resultados y metodologías de desarrollo que circulan por el mundo. En otras palabras, esto es lo mismo que decir que todos los parámetros del proceso y del producto serían crecientemente sensibles con el grado de borrosidad.

Fox, en su estimulante libro, desarrolla algo así como un primer intento serio de tipificación del software desde la óptica de su proceso. Expresa el grado de borrosidad por tres atributos: escala, complejidad y claridad. Utiliza ejemplos muy sencillos para transmitir sus ideas, como la mencionada analogía de los puentes (fig. 1) para el atributo "escala", pero no conduce sus conclusiones a establecer la singularidad de la primera fase (requisitos, diseño preliminar o especificaciones) y su impacto general en el proceso.

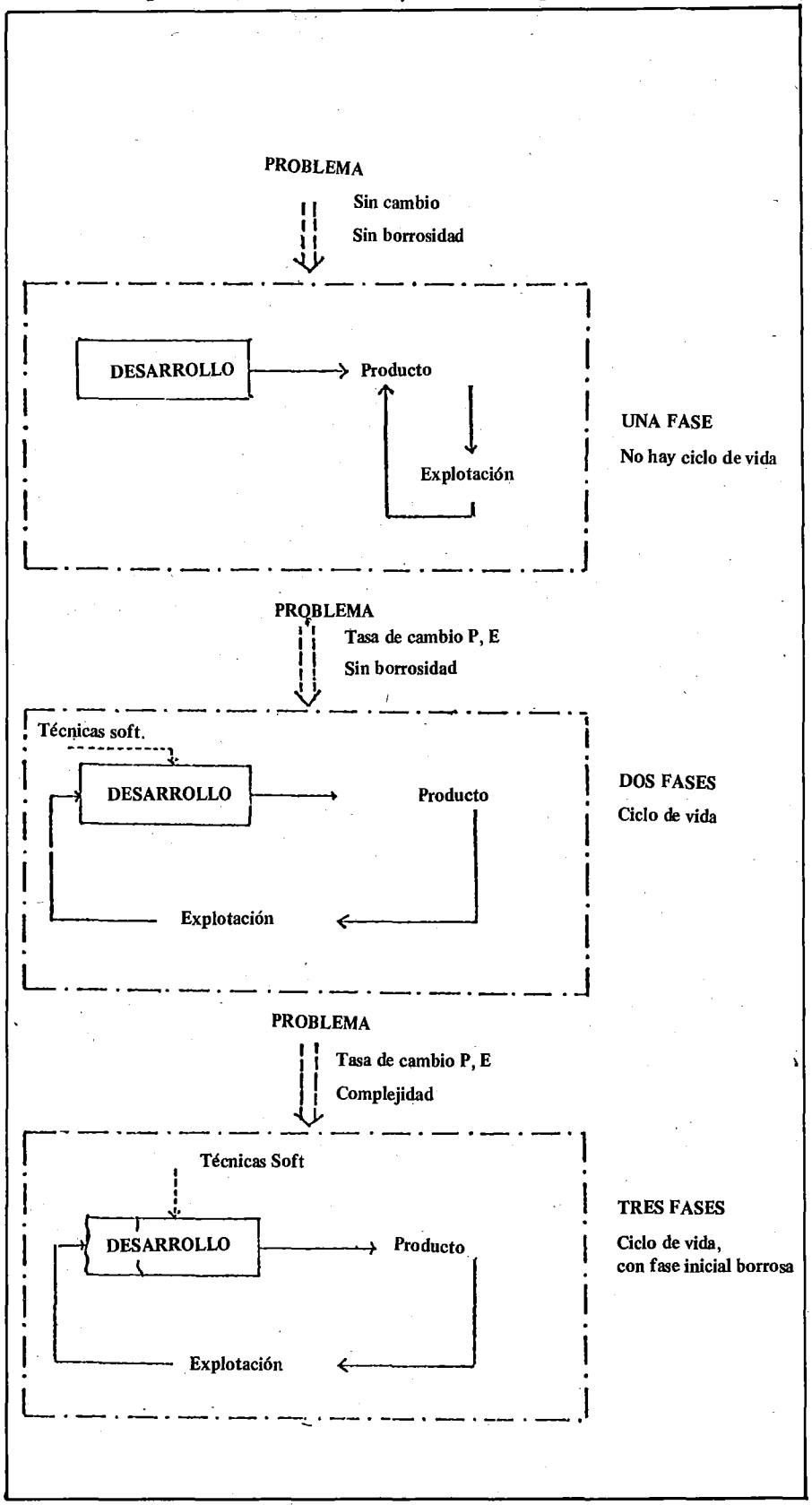
Hemos llegado a la segunda aproximación. Cuando la tasa de cambio y el grado de borrosidad son apreciables, el proceso del software debe ser diseñado como una secuencia de tres fases singulares, pero no independientes, ya que todas conforman conjuntamente su perfil y la elección de técnicas. La primera fase pertene-

ce al dominio general de la complejidad de los sistemas, no al del software. La figura 8 resume gráficamente los grandes enfoques del proceso de desarrollo en una, dos y tres fases, según los atributos aquí considerados acerca de la naturaleza del problema.

3. ESTRUCTURA INTERNA DEL PROCESO

Tropezamos aquí con un punto muy confuso. De acuerdo con nuestra figura 2, aunque aceptemos el desglose del ciclo de vida a la manera de la figura 3 o una parecida, las etapas del desarrollo deberían recibir distintos énfasis, ocasionar esfuerzos variados, emplear técnicas diversas, interrelacionarse en formas múltiples y organizarse diferentemente en el tiempo, de acuerdo, además de con las características ya analizadas del problema (u otras), con las características definitorias del producto: facilidad de uso, recuperabilidad, fiabilidad, empleo de recursos, transportabilidad,... Son notorias las diferencias en especificación entre piezas de software de aplicaciones, de sistemas o de soporte o entre un software-producto un software para un solo usuario. No sólo las características técnicas mencionadas sino los niveles de documentación, de coste y de tiempo se abren en un amplísimo abanico.

Fig. 8. Resumen de enfoques del proceso de desarrollo (influencias de la naturaleza del problema).



A mi entender, es necesario empezar a dejar de considerar el software como un ente simple y uniforme, al que se puede denotar tranquilamente por una sola palabra. Es muy cierto que se diseña y construye por procesos mentales, más aún cuando dichos procesos fueran siempre unos básicos y los mismos, en primer lugar se desconoce cuáles puedan ser éstos y, después, quedaría el infatigable problema de organizarlos en una acción integrada y siempre distinta. La ingeniería del software tiene que partir de este principio de humildad.

Hasta ahora, los más acreditados autores nos han obsequiado con sus resultados y observaciones, sin enseñar la baraja por si había naipes marcados. Y creo que no los había, al menos en su intención. Simplemente, han generalizado con ingenuidad, porque la ingeniería del software está en sus albores.

Como muestra de la falta de clarificación en el terreno que vengo comentando, recojo en las siguientes figuras algunas curvas de esfuerzo (coste) publicadas en la bibliografía. La figura 9 corresponde a la curva de Rayleigh, muy divulgada en los últimos años (Putnam, 1979).

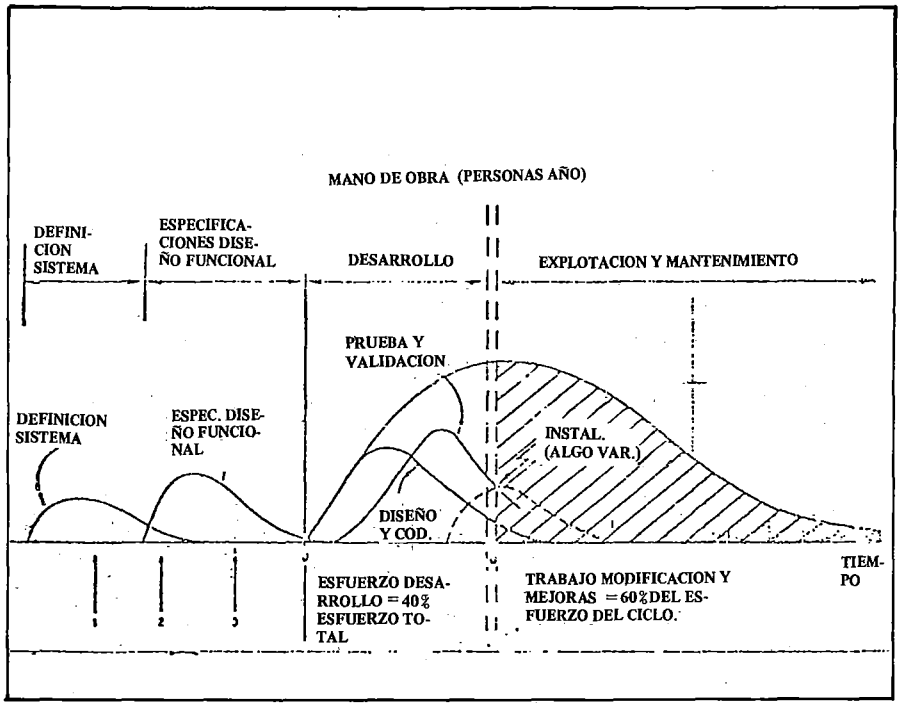


Fig. 9. CURVA DE RAYLEIGH.

Las figuras 10 y 11 proceden del libro ya citado de Fox, aunque éste las toma de otros autores. La curva de la figura 12 se debe a Ferrentino y describe, en palabras de Fox, la distribución temporal del coste del desarrollo del software de grandes sistemas, caracterizables frecuentemente por la situación de borrosidad en la que se desarrolla la etapa de análisis y definición de requerimientos. Pero hay dos recomendaciones generales, producto de la experiencia.

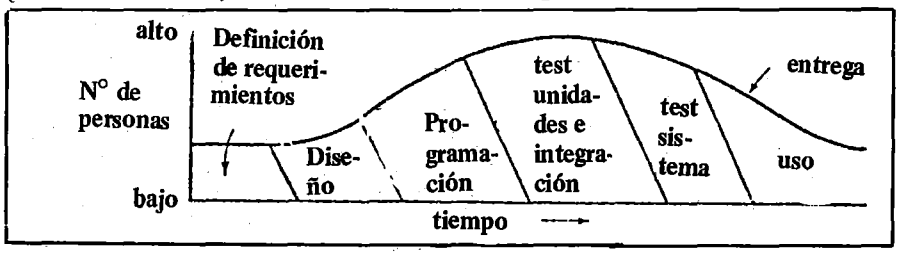


Fig. 10. Curvas de esfuerzo, hacia 1970 (FOX, 1982).

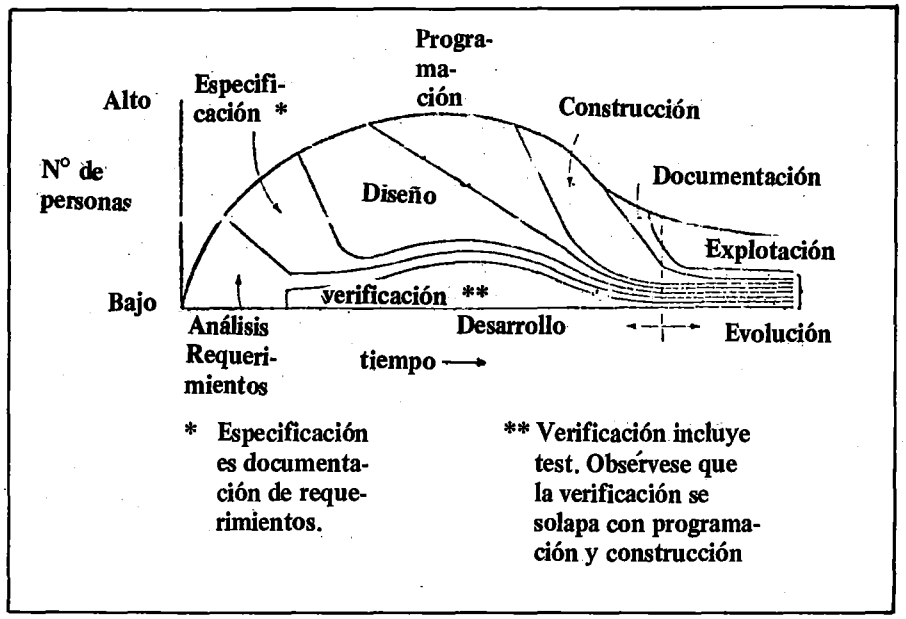


Fig. 11. Curva de esfuerzo para grandes y complejos sistemas (Fox, 1982).

1. Siendo las consecuencias globales de las primeras etapas más importantes en cuanto a la calidad final del producto, reforzar dichas etapas e investigar metodologías adecuadas.

2. Extender la actividad de test y verificación a todas las etapas del desarrollo y no sólo a la programación o integración del sistema. Una falta de coherencia en las especificaciones del sistema, descubierta a la altura de las pruebas de integración final, es en muchos órdenes de magnitud más grave que detectada en su etapa correspondiente.

Véase en la figura 12 una ilustración exacerbada (coherente y paralela con los significados de la figura 11) de la interpenetrada conectividad de las etapas del desarrollo.

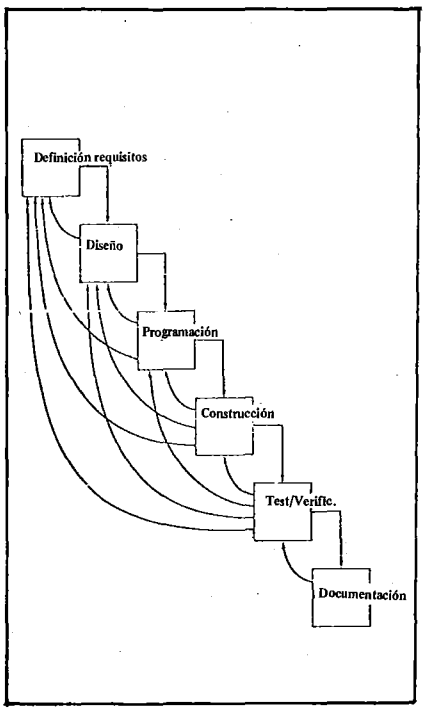


Fig. 12. Interrelación de las etapas del proceso de desarrollo para grandes sistemas, según (Fox, 1982).

Ramamoorthy, en 1978, expresaba gráficamente la aplicación de las dos recomendaciones anteriores por las curvas de la figura 13. Se refería, obviamente y aunque no lo dijera, a sistemas orientados al ciclo de vida y con moderado o bajo grado de borrosidad.

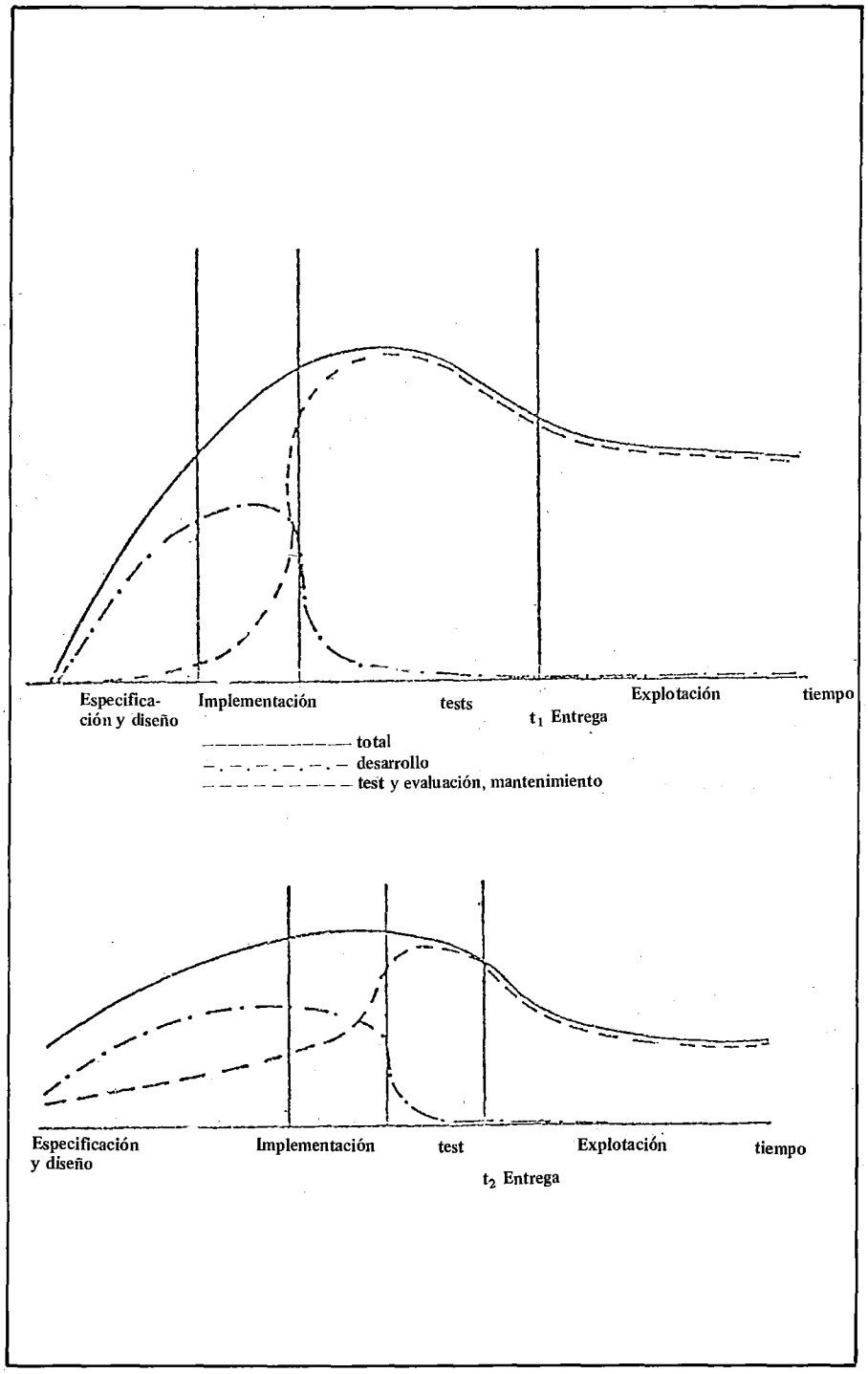


Fig. 13. Arriba: distribución convencional de costes de desarrollo del software. Abajo: distribución mejorada (Ramamoorthy, 1978).

4. INGENIERIA DEL SOFTWARE

Pese a algunas definiciones, no se sabe muy bien qué clase de ingeniería sea ésta, que, repito, no domina el producto ni el proceso que lo crea. Es un desafío, porque plantea preguntas en ámbitos excesivamente inmaduros, aunque muy ambiciosos. En todo caso, existe el movimiento que quiere llevar las cosas del estado de arte al estado de ciencia o de industria.

Aquí no se ha hablado de software, sino del proceso del software, actividad que es, al tiempo, colectiva (social), económica y técnica. Un jefe de proyecto[8] que no maneje las tres facetas no será un jefe de proyecto. Con diferente intensidad, según el proyecto, deberá ser un técnico, un gestor y un líder. No cabe duda que es una enrevesada profesión, dura y con pocos elementos de referencia para orientarse. Lo manifiesta crudamente el número de problemas que tiene planteados; según reciente encuesta, los más importantes son (Thayer, 1980):

1. Requerimientos del sistema (97%); 2. Criterios de éxito en el desarrollo del software (82%); 3. Proyecto de desarrollo (90%); 4. Estimación de costes (88%); 5. Establecimiento de calendario (94%); 6. Métodos de garantía (74%); 7. Selección de técnicas de diseño (72%); 8. Selección de técnicas de prueba (79%); 9. Estructura de contabilidad y responsabilidades (81%); 10. Selección de Jefe de Proyecto (77%); 11. Técnicas de control de fiabilidad del software (85%); 12. Técnicas de control de mantenibilidad (76%); 13. Técnicas y estándares de medida de calidad/cantidad de la producción de programadores y analistas (78%).

Los factores 1, 2, 3, 4, 5, 6, 9, 10, 11 y 13 fueron señalados unánimemente por todos los encuestados.

Un hecho notable, implícito en todas las consideraciones anteriores, es que, a medida que aumenta la relevancia de ciertos factores del problema (borrosidad (escala, complejidad), tasa de cambio) y del producto, disminuye la importancia de la única actividad central de software, la programación, y viceversa, en beneficio de las actividades del proceso. Ciertamente es así, en términos de esfuerzo y de dificultad. Piénsese que en grandes proyectos (analogía de la pasarela y del puente) las actividades de gestión superan el 50% del esfuerzo general.

¿Quiere esto decir que la programación está pasando a ser el pariente pobre de la ingeniería del software?. En términos generales, no, aunque la respuesta sea afirmativa para muchas aplicaciones. Precisamente se acaban de señalar unos cuantos problemas que son objeto de la atención prioritaria de una parte fundamental de la Ingeniería del Soft-

8. Recuérdese la escala mínima de los proyectos a los que aquí se alude.

ware, aquella ocupada en crear herramientas software para sostener y mejorar las calidades del software, el proceso en su segunda y tercera fase, si es que éstas existieran, y la propia ejecución y difusión del software.

5. EXPECTATIVAS TECNOLOGICAS

Este es un tema muy vasto, por lo que me voy a referir en solo dos pinceladas a lo que creo más importante en el contexto del artículo: los lenguajes y el entorno de programación.

El lenguaje es el instrumento con el que se materializan los procesos mentales. La variabilidad de uso de tal instrumento es increible y se pretende reducirla con vistas al desarrollo del software, entendido como actividad científica o como actividad industrial. Entre programadores se obtienen diferencias de 25 a 1 en producción de líneas de código por unidad de tiempo, de 20 a 1 en tamaño del programa, de 10 a 1 en eficiencia.

Los nuevos lenguajes de alto nivel tienden a favorecer la producción de software de calidad, a costa de un aumento de dificultad en su aprendizaje. ¿Es éste el camino? Está por ver. En todo caso, parece que ello depende mucho de cómo se vaya compensando dicha dificultad con una progresiva automatización de las ayudas en el entorno de la programación. Especial interés tiene el apoyo en exoneración de las tareas administrativas (documentación, actualizaciones, comunicación, coordinación, etc.), que puede recibir un fuerte impulso, no sólo a través del antes mencionado software-soporte (Fig. 15), sino por intermedio de un hardware diversificado y potente: microordenadores profesionales, redes locales, etc. Las funcionalidades del software y del hardware-soporte nos dirían, en cada circunstancia, las posibilidades de remodelación de los perfiles interno y externo de las curvas de la figura 10 u 11, lo que apunto en el gráfico-resumen de la figura 8 por la presencia de una flecha nominada "Técnicas soft".

Lenguaje ADA (y entorno Programación)

OBJETIVOS: Costes del ciclo de vida
Fiabilidad
Desarrollo de software portable
Desarrollo de herramientas software portables

CARACTERISTICAS: Modularización
Abstracción: Tipos abstractos de datos y compilación separada
Concurrencia: Rendez-vous (sincronización, comunicación, exclusión mutua)
Disciplina buena programación

Fig. 15. Objetivos y características del lenguaje ADA del DOD.

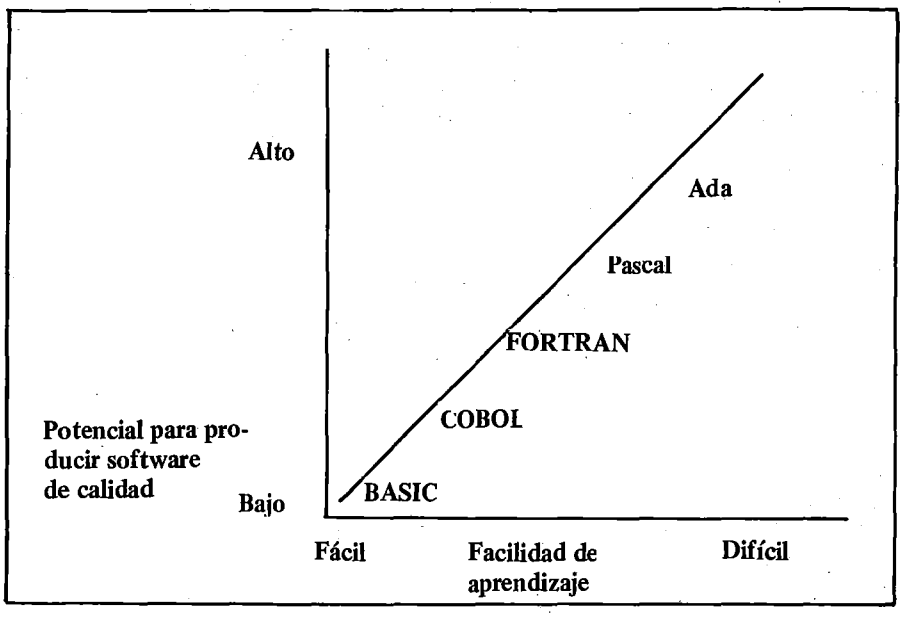


Fig. 14. Potencia del lenguaje contra facilidad de aprendizaje (Fox, 1982).

6. REFERENCIAS

1. B. Boehm, **SOFTWARE ENGINEERING ECONOMICS**, Prentice-Hall, N.J. 1981.
2. J.M. Fox, **SOFTWARE AND ITS DEVELOPMENT**, Prentice-Hall, N.J., 1982.
3. M.M. Lehman, Programs, life cycles, and laws of software evolution, **PROCEEDINGS OF THE I.E.E.E.**, Vol. 68, N° 9, Sept. 1980.
4. L. Putnam, A. Fitzsimmons, Estimating Software Cost, **DATAMATION**, Sept. Oct. y Nov. 1979.
5. C.V. Ramamoorthy, H.H. So, **SOFTWARE REQUIREMENTS AND SPECIFICATIONS: STATUS AND PERSPECTIVES**, I.E.E.E., 1978
6. F. Saéz Vacas, **ESTADO DE LAS TECNICAS DE EVALUACION DE PROYECTOS DE SOFTWARE (INGENIERIA DEL SOFTWARE EN EL CICLO DE VIDA DEL SISTEMA)**, Seminario ENTEL, 1982.
7. R. Thayer, A. Pyster, R. Wood, The Challenge of Software Engineering Project Management, **COMPUTER**, Agosto, 1980.